

PROJEKT

STEROWNIKI ROBOTÓW

Dokumentacja

Humanistycznie upośledzony robot
akrobatyczny

HURA

Skład grupy:

Albert LIS, 235534

Michał MORUŃ, 235986

Termin: sr TP15

Prowadzący:

mgr inż. Wojciech DOMSKI

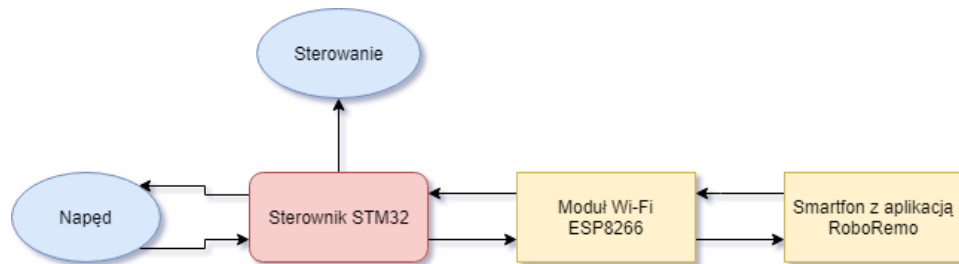
4 czerwca 2019

Spis treści

1	Opis projektu	2
2	Konfiguracja mikrokontrolera	3
2.1	Konfiguracja pinów	5
2.2	USART	5
2.3	Timer 2	5
2.4	Timer 4	5
2.5	Timer 5	6
3	Urządzenia zewnętrzne	6
3.1	Moduł Wi-Fi ESP8266	6
3.2	Smartfon z aplikacją RoboRemo	6
4	Projekt elektroniki	7
4.1	Połączenie pomiędzy STM32 a modulem WiFi	7
4.2	Schemat połączenia z serwomechanizmem	7
4.3	Schemat połączenia z silnikiem	7
4.4	Schemat połączenia z enkoderem	8
5	Konstrukcja mechaniczna	8
6	Opis działania programu	9
6.1	Schemat działania programu	9
6.2	Ręczna inicjalizacja parametrów w funkcji main	9
6.3	Funkcja obsługująca Input Capture	9
6.4	Funkcja obsługująca przerwanie USART	10
6.5	Funkcja realizująca jazdę	10
6.6	Funkcja realizująca skręcanie	10
6.7	Pętla główna	10
7	Zadania niezrealizowane	11
8	Podsumowanie	11
	Bibilografia	12

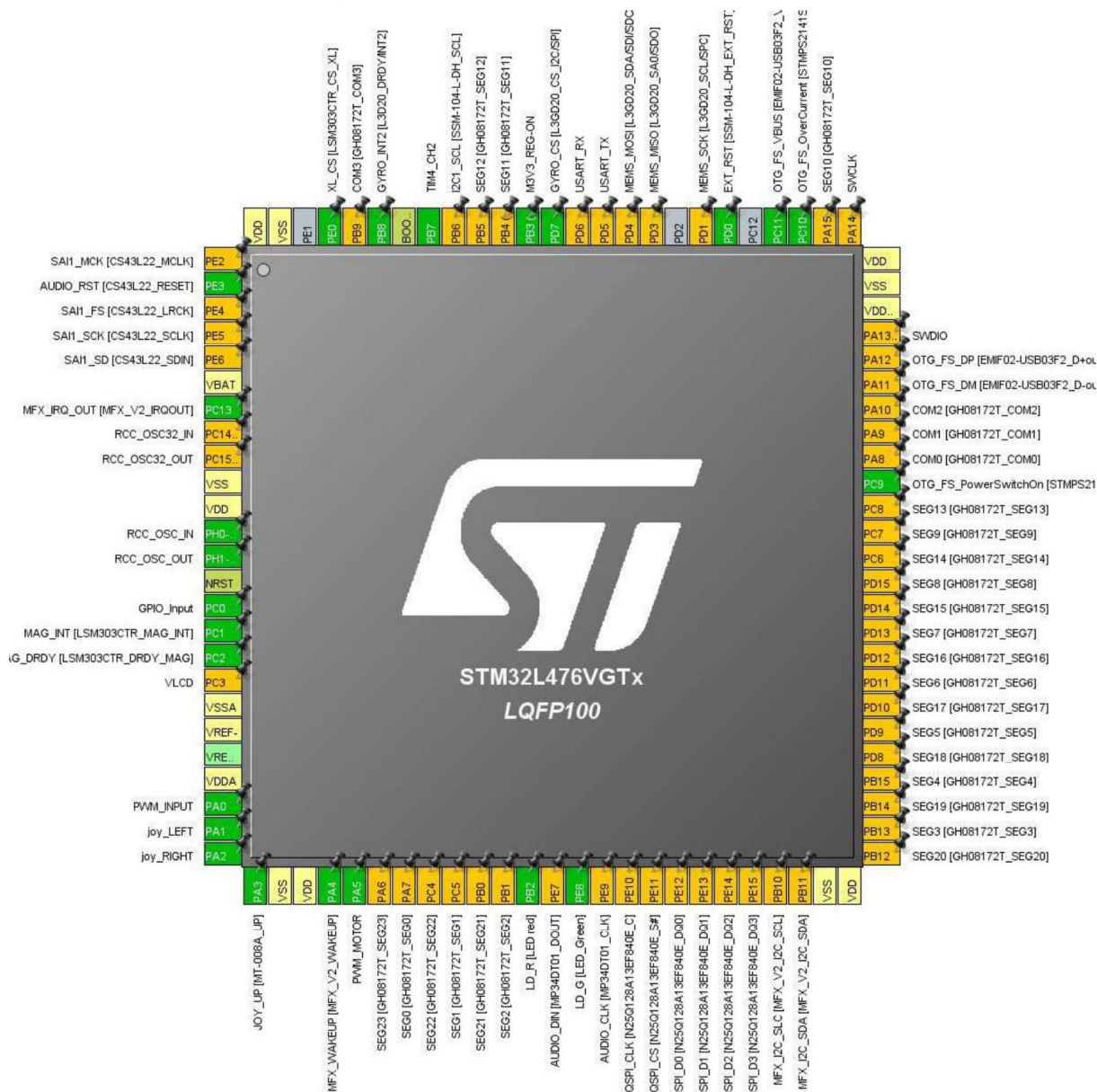
1 Opis projektu

Celem projektu jest zbudowanie zdalnie sterowanego robota jeźdnego. Robot będzie sterowany za pomocą akcelerometru w telefonie. Dane będą przesyłane za pomocą Wi-Fi lub Bluetooth. Regulacja prędkości będzie się odbywać za pomocą regulatora PID. Dane o prędkości będą pobierane z enkoderów znajdujących się w kołach robota. Opcjonalnie robot będzie wyświetlał szczegółowe dane o swoim stanie wewnętrznym za pomocą wbudowanego w płytkę z mikrokontrolerem wyświetlacza LCD.

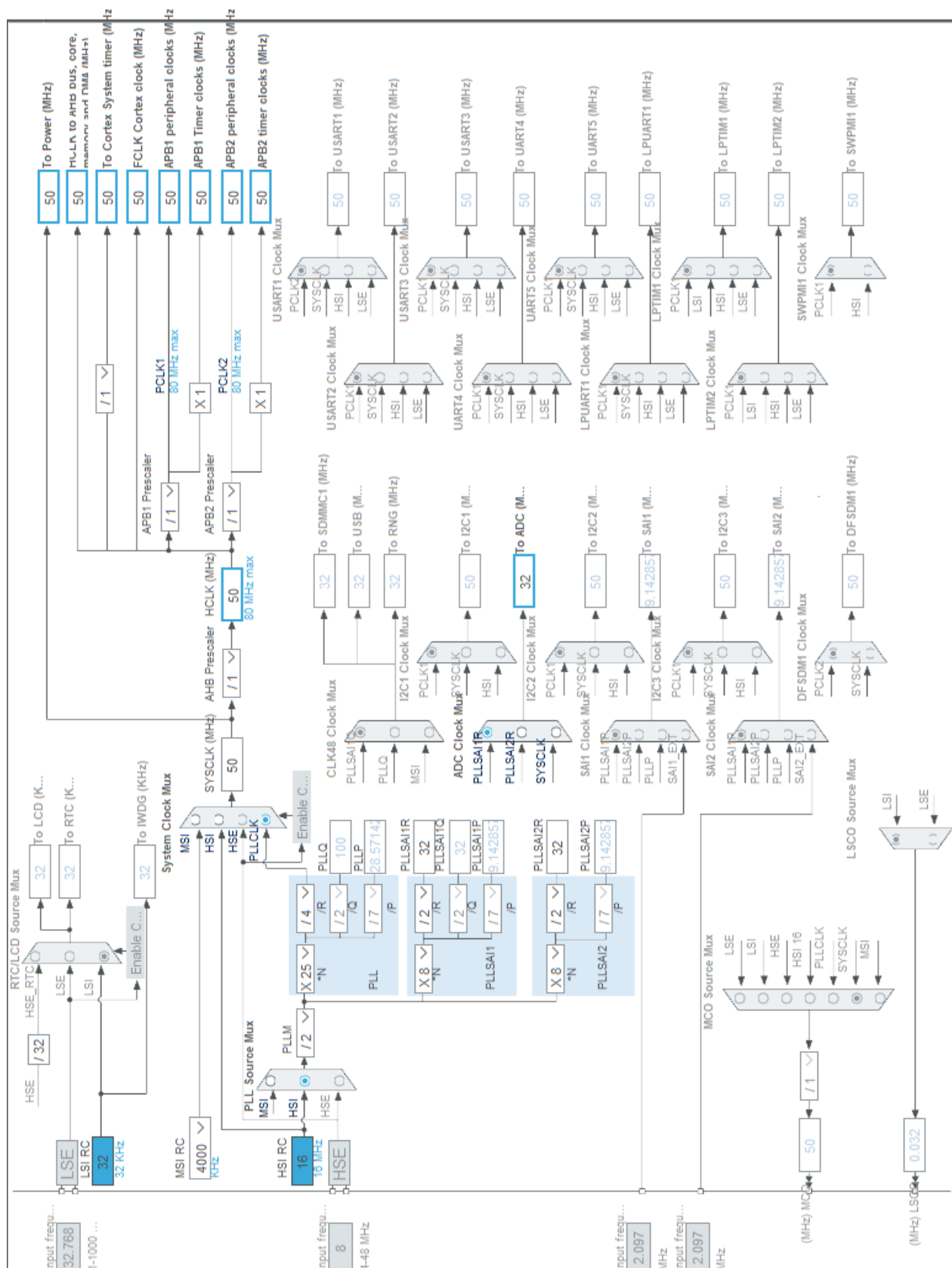


Rysunek 1: Architektura systemu

2 Konfiguracja mikrokontrolera



Rysunek 2: Konfiguracja wyjść mikrokontrolera w programie STM32CubeMX



Rysunek 3: Konfiguracja zegarów mikrokontrolera

2.1 Konfiguracja pinów

PIN	Tryb pracy	Funkcja/etykieta
PC14	OSC32_IN* RCC_OSC32_IN	RCC_OSC_OUT USART_TX USART_RX ENCODER_P DO_PRZODU SERVO
PC15	OSC32_OUT* RCC_OSC32_OUT	
PH0	OSC_IN* RCC_OSC_IN	
PH1	OSC_OUT*	
PA2	USART2_TX	
PA3	USART2_RX	
PA0	TIM5_CH1	
PA5	TIM2_CH1	
PB6	TIM4_CH1	

Tabela 1: Konfiguracja pinów mikrokontrolera

2.2 USART

Interfejs jest wykorzystywany do komunikacji z modulem Wi-Fi (ESP8266). Moduł odbiera dane za pomocą interfejsu UDP i przekazuje je do mikrokontrolera STM32 za pomocą interfejsu komunikacji szeregowej.

Parametr	Wartość
Baud Rate	115200
Word Length	8 Bits (including parity)
Parity	None
Stop Bits	1

Tabela 2: Konfiguracja peryferium USART

2.3 Timer 2

Parametr	Wartość
Clock Source	Internal Clock
Channel1	PWM Generation CH1
Prescaler	TIM2_PRESC
Counter Mode	Up
Counter Period	TIM2_PERIOD
Internal Clock Division	No Division
Mode	PWM mode 1
CH Polarity	High

Tabela 3: Konfiguracja peryferium Timer 2

2.4 Timer 4

Parametr	Wartość
Clock Source	Internal Clock
Channel	PWM Generation CH1
Prescaler	TIM4_PRESC
Counter Mode	Up
Counter Period	TIM4_PERIOD
Internal Clock Division	No Division
Mode	PWM mode 1
CH Polarity	High

Tabela 4: Konfiguracja peryferium Timer 4

2.5 Timer 5

Parametr	Wartość
Clock Source	Internal Clock
Channel	Input Capture direct mode
Prescaler	TIM5_PRESC
Counter Mode	Up
Counter Period	TIM5_PERIOD
Internal Clock Division	No Division

Tabela 5: Konfiguracja peryferium Timer 6

3 Urządzenia zewnętrzne

3.1 Moduł Wi-Fi ESP8266

Moduł dobiera dane za pomocą protokołu UDP i przekazuje je odpowiednio sformatowane za pomocą portu szeregowego do mikrokontrolera STM32. Do oprogramowania modułu wykorzystano framework Arduino.

Ustawienia komunikacji szeregowej:

Parametr	Wartość
Baud Rate	115200
Word Length	8 Bits (including parity)
Parity	None
Stop Bits	1

Tabela 6: Konfiguracja peryferium UART w module Wi-Fi

3.2 Smartfon z aplikacją RoboRemo

Przesyła wartości osi X i Y żyroskopu za pomocą protokołu UDP.
Ustawienia żyroskopu w aplikacji:

Oś X	
Parametr	Wartość
Etykieta	x
Wzmocnienie	1.0
Typ danych	int
Zakres	od -255 do 255
Limit do [min:max]	tak

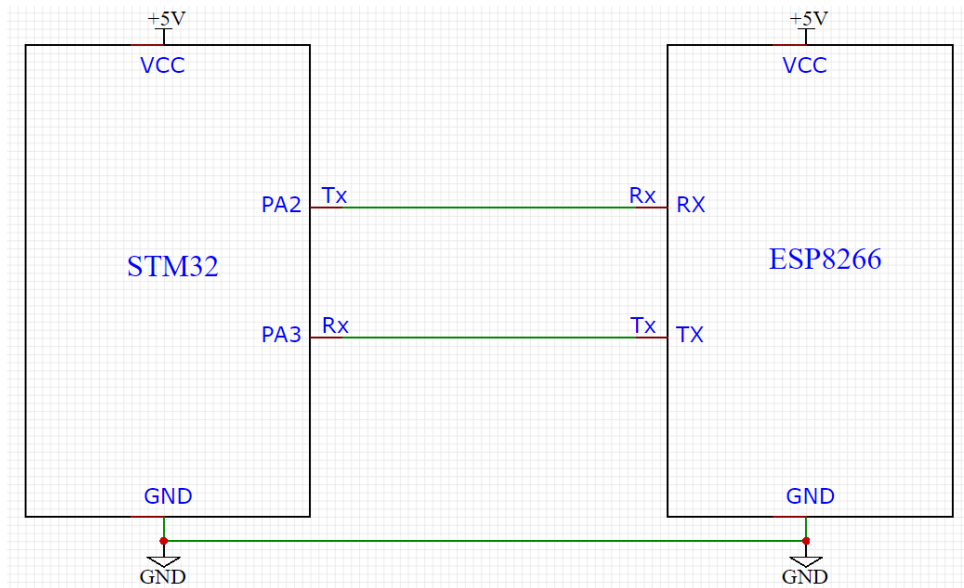
Tabela 7: Konfiguracja osi X

Oś Y	
Parametr	Wartość
Etykieta	y
Wzmocnienie	0.5
Typ danych	int
Zakres	od 50 do 150
Limit do [min:max]	tak

Tabela 8: Konfiguracja osi Y

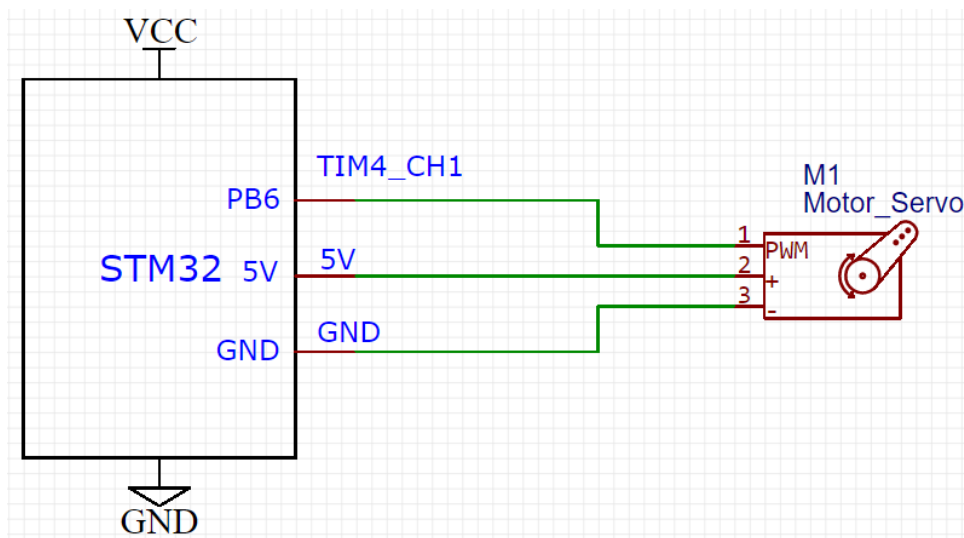
4 Projekt elektroniki

4.1 Połączenie pomiędzy STM32 a modulem WiFi



Rysunek 4: Połączenie STM32 z ESP8266

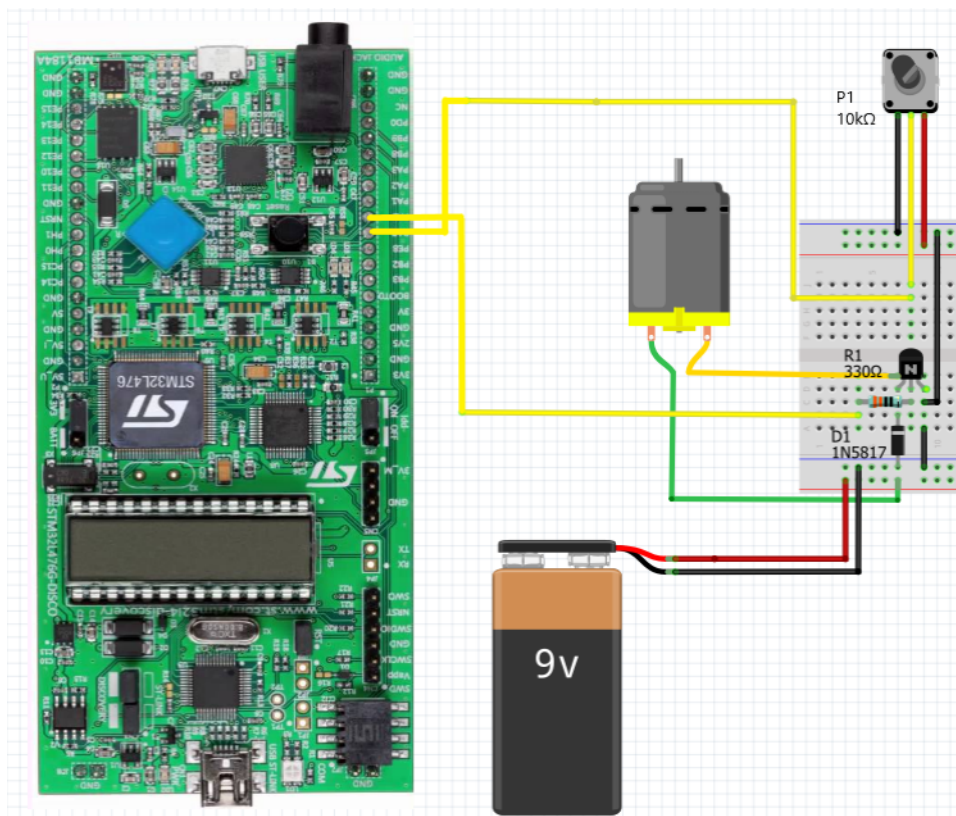
4.2 Schemat połączenia z serwomechanizmem



Rysunek 5: Połączenie STM32 z serwomechanizmem

4.3 Schemat połączenia z silnikiem

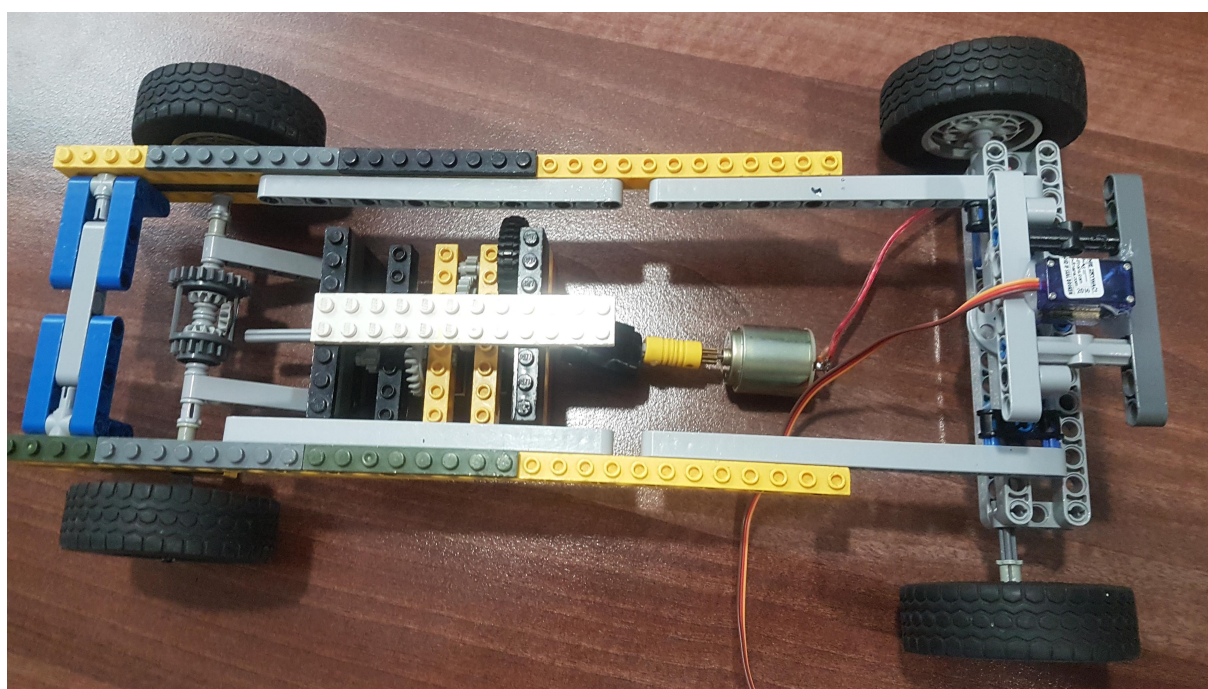
Za pomocą potencjometru regulujemy wypełnienie sygnału PWM. Sygnał ten jest wzmacniany za pomocą tranzystora NPN i przekazywany do silnika DC.



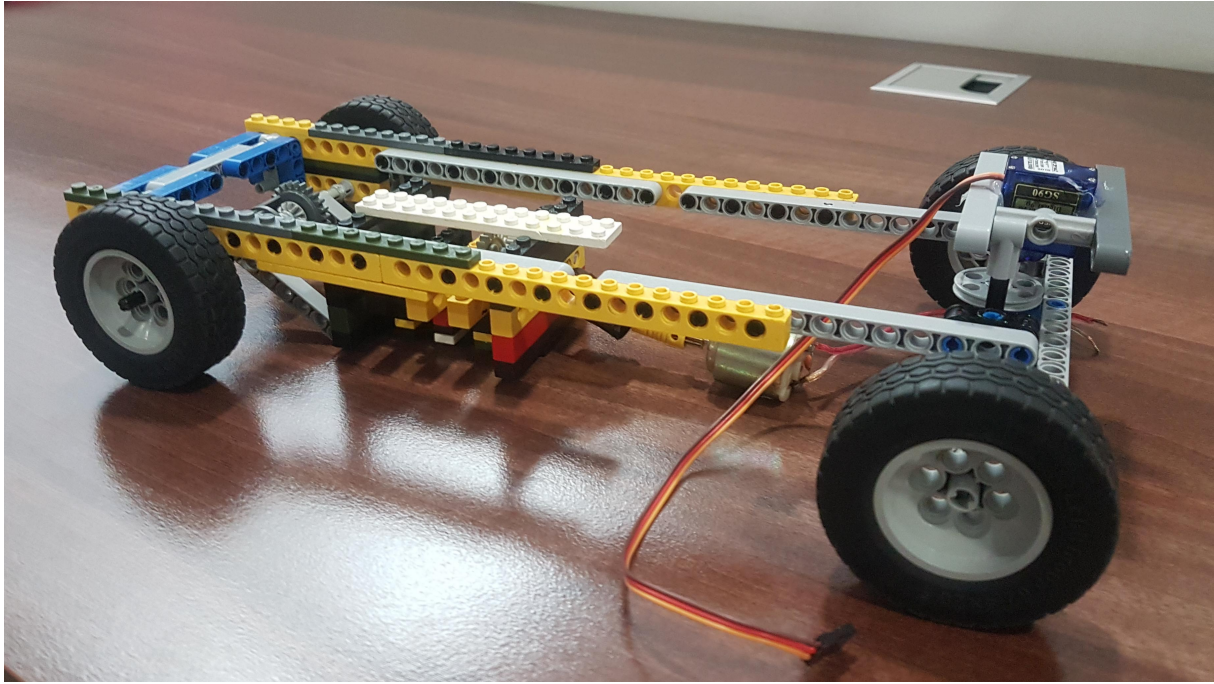
Rysunek 6: Schemat poglądowy regulacji prędkości obrotowej silnika

4.4 Schemat połączenia z enkoderem

5 Konstrukcja mechaniczna



Rysunek 7: Zdjęcie części mechanicznej nr 1



Rysunek 8: Zdjęcie części mechanicznej nr 2

6 Opis działania programu

6.1 Schemat działania programu

Rysunek 9: Schemat działania programu

6.2 Ręczna inicjalizacja parametrów w funkcji main

```

1  flag1 = flag2 = 0;
2  pwm_dutyL = pwm_dutyP = 0;
3  HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_1);
4  HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_2);
5
6  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
7  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
8  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
9
10 HAL_UART_Receive_DMA(&huart2, &Received, BUFSIZE);
11 PIDInit(&pid, kp, ki, kd, sampleTimeSeconds, minOutput, maxOutput, mode, control
    );
12 PIDSetpointSet(&pid, set_value);
13 PIDInputSet(&pid, speed);

```

6.3 Funkcja obsługująca Input Capture

```

1  int speed;
2  void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
3  {
4      if (htim == &htim5)
5      {
6          if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
7          {
8              period1 = __HAL_TIM_GET_COMPARE(&htim5, TIM_CHANNEL_1);

```

```

9      __HAL_TIM_SET_COUNTER(&htim5, 0);
10     flag1 = 1;
11     speed = (period1 <= 50) ? (255 - map(period1, 20, 50, 0, 255)) : 0;
12 }
13 sampleTimeSeconds = period1 / 1000.0;
14 PIDInit(&pid, kp, ki, kd, sampleTimeSeconds, minOutput, maxOutput, mode,
        control);
15 }
16 }

```

6.4 Funkcja obsługująca przerwanie USART

```

1  volatile uint8_t xvalue, yvalue, yprevious, xprevious;
2  volatile char xsign;
3  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
4  {
5      axis = (char)(Received[nrAxis]);
6      sign = (char)(Received[nrSign]);
7      value = (uint8_t)(Received[nrValue]);
8
9      if(axis == 'x')
10     {
11         xsign = sign;
12         xprevious = xvalue;
13         xvalue = value;
14     }
15
16     if(axis == 'y')
17     {
18         yprevious = yvalue;
19         yvalue = value;
20         if(abs(value - xprevious) < 2)
21             yvalue = yprevious;
22         else if(value > 100)
23             yvalue = yprevious;
24         else if(value < 50)
25             yvalue = yprevious;
26     }
27     HAL_UART_Receive_DMA(&huart2, &Received, BUFSIZE);
28 }

```

6.5 Funkcja realizująca jazdę

```

1  void Jazda()
2  {
3      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 255 - pid_output);
4  }

```

6.6 Funkcja realizująca skręcanie

```

1  void Skrecanie()
2  {
3      __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, yvalue);
4  }

```

6.7 Pętla główna

```

1  while (1)
2  {
3      Jazda();
4      Skrecanie();
5  }

```

- 7 Zadania niezrealizowane
- 8 Podsumowanie

Literatura